METHOD AND SYSTEM FOR MANAGING EVENTS

Field of Invention

The present invention relates to a method and system for managing events. More particularly, but not exclusively, the present invention relates to a method and system for correlating generic events from a multitude of sources.

Background to the Invention

In the world of e-business, time is collapsing as business processes involving consumers, partners, suppliers and employees operate in real-time across high-speed intranets and the Internet. Business success relies upon highly available systems and the customers' Internet experience. IT managers and CIOs are faced with constantly changing technology, ever-increasing pressures to deliver, a shortage of necessary people and skills, and the ongoing difficulty in achieving alignment between IT management and overall business objectives.

What is required is a distributed large-scale management solution that can monitor, control and report the health of the IT environment across boundaries.

On the distributed systems of the present there are implemented numerous, often disparate, hardware and software solutions. These implemented solutions generate large numbers of events, including: errors, status messages, performance variables, and customer variables. For example applications can create log files, database systems can generate error or status messages, operating systems can generate messages, MICROSOFT™ WINDOWS™ can generate event logs and performance monitors, and networks can generate SNMP (Simple Network Management Protocol) events.

In order to assist the management of these systems what is desired is a system for co-ordinating and consolidating the multitude of events from across the distributed systems.

To address the problem of managing large numbers of events from different sources two approaches have been taken:

1) Event flow model. The event flow model while very flexible tends to become very

complex to implement and requires the end user to learn a language and to also have design skills.

2) A rules engine. The rules engine also requires learning of a language and structurally becomes tremendously complicated when describing complex rules.

It is an object of the present invention to provide a method and system for managing events from a multitude of sources which overcomes the disadvantages of the above prior art and meets the requirements of real world distributed systems, or at least provides the public with a useful choice.

Summary of the Invention

According to the first aspect of the invention there is provided a method of managing events in an event engine including the steps of:

i. inputting an event into the engine;
ii. the engine extracting a rule from a rules database wherein identification information within the rule identifies the event;
iii. the engine holding the event for the expiration of a specified interval;
iv. before the expiration of the specified interval inputting a further event into the engine;
v. the engine identifying the further event using identification information within the rule;
vi. the engine creating and outputting a new event;
vii. inputting the new event into the engine; and
viii. the engine extracting a second rule from a rules database wherein identification information within the second rule identifies the new event.

The source of the event and the further event may be a network, an application, an operating system, or hardware. Preferably, intelligent agents collect the events originating from the source and transmit them on to the engine. In a preferred embodiment of the method the intelligent agents convert the events collected from the source into a common format before transmitting them on to the engine.

The identification information may include an attribute, a operator, and a value.

Preferably, the specified interval is time.

According to a further aspect of the invention there is provided a method of managing events in an event engine including the steps of:

i. inputting an event into the engine;
ii. the engine extracting a rule from a rules database wherein identification information within the rule identifies the event;
iii. the engine creating and outputting a new event;
iv. inputting the new event into the engine;
v. the engine extracting a second rule from the rules database wherein identification information within the second rule identifies the new event;
vi. the engine holding the new event for the expiration of a specified interval;
vii. before the expiration of the specified interval inputting a further event into the engine;
viii. the engine identifying the further event using identification information within the second rule; and
ix. the engine creating and outputting a further new event.

Preferably, the outputted further new event is received by a user console.

According to a further aspect of the invention there is provided a method of managing events in an event engine including the steps of:

i. inputting an event into the engine
ii. the engine extracting a first rule from a rules database wherein identification information within the first rule identifies the event;
iii. the engine holding the event for the expiration of a specified interval;
iv. before the expiration of the specified interval inputting a further event into the event engine;
v. the engine extracting a second rule from the rules database wherein identification information within the second rule identifies the further event;
vi. the engine creating and outputting a new event;
vii. before the expiration of the specified interval inputting the new event into the engine;
viii. the engine identifying the new event using identification information within the first rule; and
ix. the engine creating and outputting a further new event.

According to a further aspect of the invention there is provided a method of managing events including the steps of:

i. receiving an event;

ii. extracting a rule from a rules database wherein identification information within the rule identifies the event;

iii. when specified within the rule performing one of:

   a) creating a new event; or

   b) holding the event;

   wherein during the method at least one rule specifies performance of step a) and at least one rule specifies performance of step b); and

iv. *repeating steps i. to iii. at least once;*

   wherein at least one received event in step i. is a new event created in step iii. a).

According to a further aspect of the invention there is provided a method of generating an event in an event engine based upon two or more received events and an event previously generated by the event engine wherein at least one of the events is held by the event engine until the expiration of a specified interval.

According to a further aspect of the invention there is provided a method of managing events including the steps of:

i. processing an event by:

   a) receiving the event;

   b) extracting one or more rules which match the event from a rules database;

   c) discarding the event if at least one of the rules specifies that the event is to be discarded;

   d) holding the event if at least one of the rules specifies that the event is to be held for a period of time;

   e) altering the event or creating a new event if at least one of the rules specifies that the event is to be altered or a new event created; and

   f) outputting the event if all rules specify that the event is to be outputted;

   wherein if the event is discarded then neither of steps (d) and (e) will proceed;

ii. holding the event for the longest period of time specified by the rules if the event is specified to be held; and

iii. repeating step (i) if the event was held in step (ii).

According to a further aspect of the invention there is provided a system for managing events including:

i. a plurality of event agents adapted to receive data from a source, to create an event from the data and to transmit the event to a central event system; and

ii. a central event system including:

a) a rules database adapted to store a plurality of rules, each rule including:

I. identification information specifying to which events the rule relates; and

II. an action wherein the action is one of outputting the event, discarding the event, holding the event, or creating a new event;

wherein, where the action is holding the event the rule further includes:

I. a condition; and

II. a further action wherein the further action is one of outputting the event, discarding the event, holding the event, creating a new event, or creating a new event and transmitting the new event back into the processing engine; and

b) a processing engine adapted to receive events, to extract rules from the rules database, to identify which rules apply to the events using the identification information within the rule, to perform the action specified within the applicable rules, and to perform the further action specified within the applicable rules when the corresponding condition is satisfied.

Preferably the system also includes one or more user consoles which are adapted to receive events outputted by the central event system.

Brief D scription of the Drawings

Embodiments of the invention will now be described, by way of example only, with reference to the accompanying drawings in which:

Figure 1: illustrates the system.

Figure 2: illustrates a preferred embodiment of the engine within the central management server.

Figure 3: illustrates the structure of a rule used within the engine.

Figure 4: illustrates example 1.

Figure 5: illustrates example 2.

Figure 6: illustrates example 3.

Figure 7: illustrates example 4.

Figure 8: illustrates example 5.

Detailed Description of Preferred Embodiments

The invention will be described in relation to Figure 1.

The invention can provide a single-pane-of-glass view for close and efficient control of events happening across all systems, creating a "mission control" centre for an entire distributed environment. It monitors, filters, correlates and responds to the thousands of events that occur daily from network devices, systems, databases and applications.

Fully integrated operations and performance agents 1 (intelligent agents) provide functionality to efficiently monitor the health and performance of virtually any system.

The invention is capable of managing industry leading applications 2, databases 3, and every major operating system 4, including HP-UX, Sun Solaris, Microsoft Windows®, Linux, IBM AIX and Compaq Tru64.

The independent intelligent agents 1 provide secure and reliable communication mechanisms, advanced local filtering and corrective actions for proactive management. Flexible management concepts allow the definition of sophisticated management hierarchies, and a powerful role-based user concept supports scaling to any size. The invention is capable of managing mission-critical environments with tens of thousands of elements.

In a preferred implementation of the invention a common console 5 is provided which generates a consolidated view across all managed components, giving management staff immediate and consistent access to the status of mission-critical application services.

The invention is capable of collecting events from:

- application and system logfiles 6
- system messages 7
- customer variables
- MS Windows event log
- MS Windows performance monitor
- SNMP traps & variables 8
- MPE/iX console messages
- database status/error messages
- hardware status/error messages

The invention can perform the following operations with events:

- event processing
- event filtering, prioritizing, and grouping of messages
- sophisticated event correlation

The invention can as a result of the above operations:

- buffer messages if the management system is down
- forward messages to pre-defined systems
- perform automatic actions

The invention provides a consistent system- and fault-management process and workflow. It enables operators to use common techniques for all managed multi-vendor objects across the environment.

The preferred components of the invention are:

1. Intelligent Agents

Intelligent agents 1 can detect any failure and performance degradation of virtually any source on the managed system. They can monitor system and application logfiles 6, general system messages 7, SNMP traps and variables 8 (from networks 11), hardware components 12 (such as disks and CPUs) and customer variables from any application.

Events are converted into a standard internal format and forwarded 13 to the central management server 14.

Local buffering guarantees that all events are collected, even if the network connection to the central management server is down.

2. Central Management Server

Events are received from the intelligent agents 1. Although it will be appreciated that intelligent agents are not necessary and that events may come directly from the source.

Irrelevant and duplicated events may be suppressed (filtered out) if desired and stored in a central repository or deleted. Events can trigger pre-defined automatic actions, including the sending of messages to the user console. Processing also includes adding important or critical status information and grouping events into categories such as "security" or "OS." Using the built-in notification service, events can be automatically forwarded to other applications—for example, to flash a light or to activate a pager.

The invention provides that administrators may customise that way that events are processed and filtered by the central management server 14. Efficient event management helps to forward only relevant events to the user console 5:

• Irrelevant and duplicated events can be filtered out, or stored in a central repository.

• Optional message counters consolidate events. The user sees an event only once, including the number of occurrences instead of getting it multiple times.

• Messages can easily be correlated. For example, a "database up" message can automatically acknowledge a "database down" message.

• Custom attributes allow the extension of messages by adding any additional information, such as customer names or support levels, to the messages.

• Messages can be grouped in any way, based on message attributes.

• Service hours based on event attributes, such as time and managed node, guarantee that a user receives only messages related to services as defined.

• Outage definitions based on event attributes, such as time and managed node, prevent the user from receiving hundreds of messages that result from a planned maintenance downtime of a system, database or application.

3. User Console

Event data (messages), received from the central management server 14, are presented to the user(s) in a consistent format, completely independent from the originating source:

- Color coding (six different severity states) clearly indicates the severity of a failure or performance degradation.

- The user can drill down to information about available actions and annotations attached to a message.

- Event-specific instructions guide the user through the problem resolution process to quickly resolve a problem.

- Using interactive troubleshooting and problem resolution, users can initiate pre defined actions with a single mouse-click to fix a problem or to gather additional data. All information resulting from the action execution is stored in a central database to automate the resolution of problems over time. Users also can own and acknowledge events or forward them on (escalate them) to other operators and applications.

In a preferred implementation of the invention a user interface is provided which combines the concepts of the invention with the familiar MICROSOFT™ WINDOWS™-type concepts to minimize training time and to reduce users' learning curve. It provides a single-pane-of-glass view across your environment, integrating information from numerous sources into a single operations centre.

Features of the user console 5 include:

- The console provides all information at a glance. The core objects presented to the operator are managed nodes, available tools, message groups, and multiple message browsers.
- An intuitive graphical user interface includes a menu bar, short-cut bar and context-sensitive menus for quick and easy problem analysis and resolution.
- Customizable, reloadable views provide personal views for each operator.
- Multiple event filter browsers help the operator to concentrate on emerging and business-critical problems first. Events can be filtered and sorted by using any of the event attributes, such as the timestamp, the severity and the logical group of a message.
- Graphical chart summaries allow your operators to see the health of a system, database or application at a glance. They can be easily created using event filters.
- Pre-integrated solutions, including HP OpenView Service Navigator, Network Node Manager, Performance Manager, Problem Diagnosis and Internet Services, allow for fast problem isolation and resolution.
- The open interface allows any URL-based application to be launched from within the Java UI, using the context of an event (such as the node name).

Advanced security

In an e-business environment a management solution needs to be security aware and not create any additional exposure through its operation. Providing secure communication

mechanisms for managing business-critical and sensitive IT environments over potentially insecure network infrastructures is key to the success of an enterprise's security strategy. The invention comes with standard protection against passive attacks (eavesdropping) by securing all network traffic between the central management console and the distributed intelligent agents. The invention may provide an extended communication infrastructure to support authentication, data encryption and integrity of management data. The invention may provide data protection for the communication channels between the central management servers, distributed intelligent agents and the Java user interfaces.

Event reduction and consolidation

In addition to event filtering, the invention provides a method to solve the well known challenges of event reduction and consolidation across networks, systems, databases and applications. The significant reduction of the event load allows IT staff to manage a larger environment with the same resources. The invention provides correlation capabilities on supported intelligent agent platforms and on the central management server correlating data from any combination of event sources. The invention may provide for the development of correlation rules using an interactive graphical point-and-click user interface. Using a unique simulation mode, the correlation logic can be tested before it is deployed to the intelligent agents.

Preferably the invention is implemented as follows:

The central management server on any of:
• HP-UX™ 11.0, 11.11 or Sun Solaris™ 2.7, 8; Sun Cluster™ 3.0
• Oracle™ 8.1.7 Enterprise Edition (32-Bit, 64-Bit)
• Oracle™ 9.0.1 Enterprise Edition (64-Bit)

The user console in Java on any of:
• HP-UX™ 11.0, 11.11
• Sun Solaris™ 7, 8
• Microsoft Windows NT™, Windows 2000™, Windows 98™
• JRE 1.3.1 or higher for running the UI as a Java application on MS Windows, JRE 1.3.2 on HP-UX™ and Sun Solaris™
• Java plug-in 1.3.1 for running it in Internet Explorer™ 5.0, 5.5 and Netscape Navigator™ 4.7, 6.1 on Microsoft Windows NT™, Windows 2000™ and Windows 98™

The intelligent agents on any of:

- HP-UX™ 10.20, 11.0, 11.11, 11.22
- Sun SPARC Solaris™ 2.6, 7, 8, 9; Sun Cluster™ 3.0
- Microsoft Windows NT™ 4.0, Windows 2000™ 5.0, Windows XP™ (32 bit)
- IBM RS/600 AIX 4.3.1, 4.3.2, 4.3.3, 5.1
- Compaq Tru64™ UNIX 4.0F, 4.0G, 5.0A, 5.1, 5.1A, 5.1B
- Tru64 Cluster™ 5.1A, 5.1B
- Red Hat™ Linux 6.2, 7.0, 7.1, 7.2, 7.3
- SuSe™ Linux 6.2, 6.3, 6.4, 7.0, 7.1, 7.2, 7.3, 8.0
- Turbo Linux 6.0J, 6.1J, 6.5J, 7.0J
- Debian™ Linux 2.2r3, 2.2r4, 2.2r5
- OS/390, OS/400

It will be appreciated by those skilled in the art that the central management server may be deployed on any computer system. It will be appreciated that the user console may be programmed in any language and deployed on any computer system. It will be further appreciated that the intelligent agents may be developed for any operating system.

A detailed description of an implementation of the processing engine within the central management server will now be given with reference to Figure 2.

Event Flow

An incoming event is inputted into a node 16 that evaluates the rules and determines what needs to be done with the event. There are four possibilities:

1) Take path 17 and output the event 18.
2) Take path 19 and discard the event 20.
3) Take path 21 and hold the event for a specified time 22.
4) Take path 23 and modify the event and/or create one or more new events 24.

The path to take depends on the semantics of the rule being applied to the event. The event may take more than one of the paths - for example the rule semantics may choose to send the event on all paths 17, 21, and 23.

If the rule determines that the event needs to be held then the event takes path 21. After

the specified time expires the event flows out of the "HOLD" 22 and along path 25 to a node where the rules are re-evaluated to determine the fate of the event. There are three possibilities:

1) Path 27 and output the event 18.
2) Path 28 and discard the event 20.
3) Path 29 and modify the event and/or create one or more new events 24.

As before, the event may take more than one path - the path(s) taken depends on the semantics of the rule.

Created/Modifed events can flow back 30 into the system (i.e the new events can also take part in correlation logic, if so desired).

In a preferred implementation of the invention, if the same event takes part in multiple roles the sum of the decisions of the individual rules is applied. An event is output if and only if no rule discards it or holds it.

Rules Database

The rules database 31 is a repository of rules.

A preferred structure of a rule will now be described with reference to Figure 3.

Preferably each rule 32 is a tuple of two - (alarm signature 33, functions 34). Alarm signature is an array of conditions and is a 3-tuple of the format (attribute 35, operator 36, value 37), while functions is a tuple of 2 - (input function 38, output function 39).

The idea of the alarm signature is this - if the incoming event matches all the conditions specified by the alarm signature then the rule is applied to the incoming event (Note: An event can be operated upon by multiple rules if the alarm signatures match).

When an event meets the alarm signature criteria specified then the input function is invoked and the event is passed to the input function. The return value of the input function determines which of the paths the event will take (paths in 17, 19, 21 and 23 in Figure 2). If the input function specifies that the event needs to be held, the duration is also specified. After the event is held for the specified period, the output function is

invoked. The return value of the output function determines the path that the event will take (paths 27, 28, and 29 in Figure 2).

Pseudo code for the engine is provided below:

```
Correlation_Engine
     if the incoming event is identified by a rule
     then
          call the input function for the event
          Switch on the return value of the called input
          function:
               PASS : send the event out
               ALTER/CREATE: Create a new event as per user
               specification
               HOLD: Hold the event for the specified period
               after the hold period time is over call the output
               function
               Switch on the return value of the called output
               function:
                    PASS : send the event out
                    ALTER/CREATE: Create a new event as per user
                    specification
                    DISCARD: discard the event
               end Switch
          DISCARD: discard the event
          end Switch
     else
          do nothing
     endif
end
```

Examples to illustrate the invention:

The following examples illustrate the engine used within the central management server.

Example 1

Referring to Figure 4, a first example will be described.

In this example, a "repeated correlation" rule is required that forwards 40 (outputs) the 1st event 41 of a certain type and suppresses 42 (discards) any other events 43 of the same type within a ten minute window 44. At the end of the ten minute window a new event 45 is created that indicates the number of events suppressed.

In the example the event is recognized by the following attributes:
1) Enterprise = 1.2.3.4, and
2) Specific-trap = 10 or 20, and
3) Generic-trap = 1

In the example the SNMP-TrapPDU format is used to describe the event. However, the invention itself is agnostic to the format. The invention may be configured so that incoming events in a particular format may be interpreted by the engine itself. Alternatively, a multitude of intelligent agents may received events in a multitude of formats and send the events in a standard format to the engine. For example, other formats that might be used include: SNMP, CMIP, X733 and OpC (the internal format for HEWLETT PACKARD™ - OpenView events).

To implement the rule the following steps are taken:

1) A "repeated correlation" rule is created and in the alarm signature section of the rule the following is entered:

```
[("enterprise", 'equals", 1.2.3.4), ("specific-trap", "is in the
list", [10,20]), ("generic-trap", "equals", 1)]
```

The above is an array of tuples that describes the events for which the "repeated correlation" rule is to be applied to.

2) The functions section of the rule is specified as:

```
("Repeated_Event_Input", "Repeated_Event_Output")
```

The above specifies the names of functions that need to be revoked when the event enters and when the event finishes waiting.

It will be appreciated that the functions can be implemented in any computing language.

The pseudo-code for the functions is given below:

```
RepeatedEvent_Input_Function
      if 1st event
      then
            return PASS and HOLD for 10 minutes
            //tell the engine to output the event and also hold the
            //event for 10 minutes

      else //This is not 1st event
            increment number of events discarded
            return DISCARD
            //Instruct the engine to discard the event
      endif
end

RepeatedEvent_Output_Function
      specify what the new event should look like (including the
      number of discarded events)
      return CREATE
end
```

Example 2

Referring to Figure 5, a second example will be described.

In this example, a portal service depends on the system being up, performance being good, and the network links staying up. The network generates a number of performance events. What is required is a system to correlate the failure of the portal service with performance of the network.

A first rule is configured to receive the following network events: “bandwidth utilisation”, “dropped packets” 46, “swap full”, and “proc table full”. This rule is configured to create 47 a new event when it receives any of those network events called “perf alarm”. “Perf alarm” has an attribute called *rootcause* which is set to the type of network event received (i.e. “bandwidth utilisation”, “dropped packets”, “swap full”, or “proc table full”). This new event is specified within the rule to be fed back 48 into the engine.

The alarm signature for the first rule is:

```
("event type", "is in the list", ["bandwidth utilisation",
"dropped packets", "swap full", "proc table full"])
```

A second rule is configured to receive a “portal down” event 49 from the portal service. This rule is configured to hold 50 the “portal down” event for a specified time and wait 51 for one of the following events: “perf alarm” 52, “link down”, or “system down”.

If the “perf alarm” event is received then the real root cause of the portal failing is the *rootcause* specified within the “perf alarm” event. In this example, a new event – “portal failure cause” – is created 53. An attribute within the “portal failure cause” event is set to the *rootcause* attribute of the “perf alarm” event received.

The alarm signature for the second rule is:

```
("event type", "is in the list", ["portal down", "perf alarm",
"link down", "system down"])
```

Pseudo-code to implement the input function for the first rule is provided (this rule does not have an output function as the event is not held):

```
NetworkPerformance_Input_Function
      specify new event PERF_ALARM with attribute rootcause equal
to event type
      return CREATE
end
```

Pseudo-code to implement the input and output functions for the second rule is provided:

```
PortalFailure_Input_Function
     if event is of type PORTAL_DOWN
     then
          store eventid
          return HOLD
     else
          if store has an event of type PORTAL_DOWN
          then
               if event is of type PERF_ALARM
               then
                    set portal_failure_cause as rootcause of
                    PERF_ALARM
                    return DISCARD
               else
                    set port_failure_cause as event type
                    return DISCARD
               end if
          else
               return PASS
          endif
     endif
end

PortalFailure_Output_Function
     if portal_failure_cause exists
     then
          specify new event PORTAL_FAILURE_BECAUSE with attribute
          rootcause equal to portal_failure_cause
          return CREATE
     else
          return PASS
     endif
end
```

Example 3

Referring to Figure 6, a third example will now be described.

In this example, a database system occasionally crashes but comes back on-line after a couple of minutes. The database manager only needs to be alerted to the crash if the database does not automatically restart within five minutes. Therefore a system is desired to correlate a “database down” event with a “database up” event within five minutes.

A rule is configured to receive a “database down” event 54 and to hold 55 that event for five minutes 56. The rule is further configured to wait for a “database up” event 57 within that time window 56. If a “database up” event is not received in the time window then the “database down” is sent on (output). If the “database up” event is received, then both events are correlated 58 (discarded).

The alarm signature for the rule is:

```
("event type", "is in the list", ["database down", "database up"])
```

Pseudo-code to implement the input and output functions of the rule is provided below:

```
Transient_Input_Function
     if event is of type DB DOWN
     then
          store eventid
          return HOLD
     else
          /* this is an UP event */
          if store has an event of type DB DOWN
          then
               mark down events for discard
               return DISCARD
          else
               return PASS
          endif
     endif
```

```
end

Transient_Output_Function
      if event has been marked for discard
      then
            return DISCARD
      else
            return PASS
      endif
end
```

Example 4

Referring to Figure 7, a fourth example will now be described.

In this example, packets are occasionally dropped over the network. This can lead to low latency which may be the cause of problems for other applications. Therefore a system is desired to correlate multiple instances of dropped packets to conclude whether a network has low latency.

A first rule is configured to receive a “dropped packet” event 59 and to hold 60 that event for one minute 61. The rule is further configured to wait for additional “dropped packet” events 62 within that time window. If a certain number of additional events are received the rule is configured to generate a new event 63 “low latency” with an attribute set to the number of dropped packets received, and to feed that event back 64 into the engine.

The alarm signature for the first rule is:

```
("event type", "equals", "dropped packet")
```

A second rule is configured to receive the “low latency” event 65. The rule may be configured to discard that event if for example the number of dropped packets received is within specified thresholds, or to pass on the event to a user console, or to hold the event and wait for other events which may explain the dropped packets, or to generate a new event 66 if the number of dropped packets exceeds a certain threshold.

The alarm signature for the second rule is:

```
("event type", "equals", "low latency")
```

Pseudo-code to implement the input and output functions of the first rule is provided below:

```
Packets_Input_Function
     if 1st event
     then
          return HOLD for 1 minutes

     else  //This is not 1st DROPPED PACKET event received in the
          //last minute
          increment number of dropped packets received
          return DISCARD
          //Instruct the engine to discard the event
     endif
end

Packets_Output_Function
     specify creation of new event LOW LATENCY with attribute
     dropped packets set to the number of dropped packets received
     return CREATE
end
```

Pseudo-code to implement an example of an input function for the second rule is given:

```
Latency_Input_Function
     if attribute dropped packets < 100
     then
          return DISCARD
     else
          if attribute dropped packets > 1000
```

```
            then
                  specify creation of new event EMERGENCY NETWORK
                  FAILURE
                  return CREATE
            endif
      else
            return PASS
      endif
end
```

Example 5

Referring to Figure 8, a fifth example will now be described.

In this example, there are many different ways that a server can fail. When a server fails the server manager will generally need to be notified. Applications that access the server may send out server failure messages. Once the server manager has been notified of the server going down, however, there is no need for them to receive further messages. Therefore a system is required to monitor for all types of server failure and send a message on the server manager and to block subsequent failures regarding the server.

A first rule is configured to receive "server disk failure" 67, "server processor failure", and "server memory failure" events. The rule is further configured when it receives any of the previous events to generate 68 a single "server down" event and to feed that back 69 into the engine.

The alarm signature for the first rule is:

```
("event type", "is in the list", ["server disk failure", "server
processor failure", "server memory failure"])
```

A second rule is configured to receive the "server down" event 70 to output 71 the event and to hold 72 the event for sixty minutes 73. The rule is further configured to receive within that time window 73 any "cannot connect" events 74 from applications that are failing to connect to the server, and discard them 75. The rule is further configured to

receive a “server up” event 76. If such an event is not received at the end of the time window the “server down” event is fed back into the engine.

The alarm signature for the second rule is:

```
("event type", "is in the list", ["server down", "cannot connect",
"server up"])
```

Pseudo-code to implement the input and output functions of the first rule is provided below:

```
ServerFailure_Input_Function
      specify creation of new event SERVER DOWN
      return CREATE
end
```

Pseudo-code to implement the input and output functions of the second rule is provided below:

```
BlockFailMsgs_Input_Function
      if event is of type SERVER DOWN
      then
            store eventid
            return PASS and HOLD for 10 minutes
            //tell the engine to output the event and also hold the
            //event for 60 minutes
      else
            if store has SERVER DOWN
            then
                  if event is of type SERVER UP
                  then
                        store eventide
                        return PASS
                        //The server is back up, pass this message
                        //on
```

```
                else //The event is cannot connect
                    return DISCARD
                    //Instruct the engine to discard the event
                endif
            else
                return PASS
            endif
        endif
end

RepeatedEvent_Output_Function
        if eventid not equals SERVER UP
        then
            specify new event SERVER DOWN
            return CREATE
            //If the server is not back up then feed back server
            //down event to the engine
        else
            return DISCARD
            //Server is back up
        endif
end
```

An advantage of the invention is that implementation of any kind of correlation can be done within the same engine. The output is dependant on individual rules therefore a problem can be broken up into smaller and simpler rules. The feedback mechanism allows hierarchal building of correlation models. This means that the invention is simpler to implement and maintain over event correlation/management system.

Additional advantages of the invention include:

- event filtering, consolidation and correlation;
- fast problem isolation and automatic correction; and
- a central operator console

While the present invention has been illustrated by the description of the embodiments thereof, and while the embodiments have been described in considerable detail, it is not the intention of the applicant to restrict or in any way limit the scope of the appended claims to such detail. Additional advantages and modifications will readily appear to those skilled in the art. Therefore, the invention in its broader aspects is not limited to the specific details representative apparatus and method, and illustrative examples shown and described. Accordingly, departures may be made from such details without departure from the spirit or scope of applicant's general inventive concept.